```
        list      p=16c505              ; list directive to define processor
        #include <p16c505.inc>         ; processor specific variable definitions

        __CONFIG    _CP_ON & _WDT_OFF & _MCLRE_OFF & _IntRC_OSC_CLKOUTEN

; '__CONFIG' directive is used to embed configuration word within .asm file.
; The lables following the directive are located in the respective .inc file.
; See respective data sheet for additional information on configuration word.


;***** VARIABLE DEFINITIONS
PWCount             EQU     0x08    ; counter used for output pulse width
                            ;  must count to 20 (200ms)

CountLo             EQU     0x09  ; low byte of 5 minute counter (255=2.55 seconds)
CountHi             EQU     0x0a  ; high byte of 5 minute counter     (118=5
minutes)

State       EQU     0x0b  ; bit0 high if light has been turned on
                          ; bit1 high if past initial blanking period (10 seconds)
                          ; bit2 high to prevent retriggering from shutoff pulse
                          ; bit3 high to indicate checking in progress
                          ; bit4 high to indicate RS232OK
                          ; bit5 high to indicate WORKLIGHT on
```

```
#define         LITEON      0     ;
#define         AWAKE 1     ;
#define         BLANK 2     ;
#define         CHECKING 3  ;

BlankCnt    EQU   0x0c  ; used to prevent triggering from shutoff pulse

PresCnt             EQU   0x0d  ; used to count presence of signal, 10 ms/count

PWMCount    EQU   0x0e  ; used as main counter for pwm functions
PWMVal              EQU   0x0f  ; high duration count (0 - 255)
PWMRem              EQU   0x10  ; low duration count (0 - 255)
PWMCycle    EQU   0x11  ; counts cycles, need 8 to run 10ms

IncCount    EQU   0x12  ; counts high readings before incrementing pwm
DecCount    EQU   0x13  ; counts low readings before decrementing pwm
StoredPWM   EQU   0x14  ; stored value of PWM, used to check if light has
increased or decreased
CheckCnt    EQU   0x15  ; counter used to count one second before checking light
value
LoCnt       EQU   0x16  ; counter used to measure low pulse width on line
WarmBoot    EQU   0x17  ; set to 0x55 in normal ops.  Check if==0x55 on powerup,
cold boot if not

LeapCount   EQU   0x18  ;
PlungeCount EQU   0x19  ;


;     RC0 - photocell input, high for Dark
;     RC1 - PIR input 1, low for disturbance
;     RC2 - PIR input 2, high for disturbance
;     RC3 - Pulse output, 250ms high pulse to drive transistor
;     RB2 - Test input - when low, startup timer is eliminated, and the light is
held on
;           for 3 seconds instead of 5 minutes.
#define         DARK        0
#define         PIRH        1     ; active high, pin 9
#define         PIRL        2     ; active low, pin 8
#define         PULSEOUT    3     ; transistor drive output
#define         PWMOUT              5     ; test output on RC5
#define         RS232DRV    4     ; test output on RC4
#define         TESTP13             0     ; test output on RB0
#define         WORKLIGHT   5     ; bit 5 of State byte, high to indicate that
worklight is on (based on query)
#define         RS232OK             4     ; bit 4 of State byte, indicates that
RS232 is supported by GDO


;*****************************************************************************
            ORG     0x3FF               ; processor reset vector
; Internal RC calibration value is placed at location 0x3FF by Microchip
; as a movlw k, where the k is a literal value.

        ORG     0x000               ; coding begins here
        movwf   OSCCAL              ; update register with factory cal value
        clrf    FSR                 ; ensure FSR register points to Bank0
```

```
; Setup option register for prescaling, timer uses internal clock and prescaler

;       movlw 0x0               ; temporary patchout to speed sim, @@@
        movlw 0x044             ; set prescaler to divide by 32, disable pullups
                                ; timer period is 32us
        option                  ;

; Setup ports
;       RC0 - photocell input, high for Dark
;       RC1 - PIR input 1, high for disturbance
;       RC2 - PIR input 2, low for disturbance
;       RC3 - Pulse output, 250ms high pulse to drive transistor
;       RB2 - Test input - when low, startup timer is eliminated, and the light is
held on
;             for 3 seconds instead of 5 minutes.

        movlw 0x07              ; set RC3,4,5 only as output
        tris  PORTC             ;

        movlw 0x6               ; set RBX as outputs,except for RB2 and RB1
        tris  PORTB             ;

        bcf   PORTB,5                   ; turn on power to amplifier

        bsf   State,BLANK ; set BLANK so that vacation mode won't cause retriggers

        clrf  BlankCnt    ;

        movlw 0x55
        subwf WarmBoot,w  ; if WarmBoot==0x55, assume warm boot and go to main
loop
        btfsc STATUS,Z    ;
        goto  main_loop   ;

        bcf   PORTB,TESTP13     ;

        clrf  TMR0              ; start timer off at zero
        bcf   PORTC,RS232DRV    ;
        clrf  PWCount           ; initialize all variables
        clrf  CountLo           ;
        clrf  CountHi           ;
        clrf  State       ;
        clrf  PORTC       ;
        clrf  PORTB       ;
        clrf  PresCnt           ;
        bcf   State,BLANK ;
        bcf   State,CHECKING    ;
        clrf  CheckCnt    ;
        clrf  IncCount    ;
        clrf  DecCount    ;
        clrf  LoCnt       ;
        movlw 0x7f
        movwf PWMVal            ; temporary values for sim
        movwf PWMRem            ;
        clrf  PlungeCount ;
        clrf  LeapCount   ;
```

```
main_loop

; turn on PWM output
      clrf  PWMCycle     ;
      movlw 0x07         ; set RC3,4,5 only as output
      tris  PORTC        ;

; set pwm output high
PWMStart:
      bsf   PORTC,PWMOUT      ;
      clrf  PWMCount     ;

; count PWMVal counts
PWM1:
      incf  PWMCount,1   ;
      movf  PWMVal,0     ; put PWMVal into w
      subwf PWMCount,0   ; w= PWMCount - PWMVal (if result is positive or zero, C
is set)
      btfss STATUS,C     ; if C is clear, stay in the loop
      goto  PWM1         ;
      clrf  PWMCount     ;

; clear PWM output
      bcf   PORTC,PWMOUT      ;

; count PWMRem counts
PWM2:
      incf  PWMCount,1   ;
      movf  PWMRem,0     ; put PWMRem into w
      subwf PWMCount,0   ; w = PWMCount - PWMRem
      btfss STATUS,C     ; if C is clear (PWMRem>PWMCount), stay in loop
      goto  PWM2         ;


; this point is hit about every 1.6ms
; check if line is low for three consecutive cycles - if so, go to sleep - if
not, clear counter
      btfsc PORTB,1           ;
      goto  linehi            ;
      incf  LoCnt,1           ;
      movlw 3           ;
      subwf LoCnt,0           ;
      btfss STATUS,Z    ;
      goto  chkcycles   ;
      bsf   PORTB,5           ; turn off analog section
      movlw 0x55        ;
      movwf WarmBoot    ;
      sleep             ; exit from sleep will be through reset

linehi:
      clrf  LoCnt       ;

; check if PWM program has run 6 times - if not, run it again
chkcycles:
      incf  PWMCycle,1  ;
      movlw 0x6         ;
```

```
        subwf PWMCycle,0   ;
        btfss STATUS,C     ;
        goto  PWMStart     ;

  ; if so, turn off PWM output and go to processing functions
        movlw 0x27         ; set RC3,4,5 only as output
        tris  PORTC        ;


  ; check comparator - if low, reduce output voltage
        btfsc PORTC,0              ;
        goto  boostpwm     ; if light comparator is high, go to boost output
  voltage
        clrf  LeapCount    ;
        incf  DecCount,1   ;
        movlw 0xa          ;
        subwf DecCount,0   ; check if DecCount is >10
        btfss STATUS,C     ; if not, get out of ad ops
        goto  ad_done              ;
        clrf  DecCount     ; if it is >10, clear DecCount
        movf  PWMVal,1     ; check if PWMVal is 0 - if not, decrement it
        btfsc STATUS,Z     ;
        goto  ad_done              ;
        decf  PWMVal,1     ; decrement PWMVal, put back in PWMVal
        incf  PlungeCount,1        ; increment PlungeCount
        movlw 0xc          ;
        subwf PlungeCount,w        ; check if PlungeCount>12 -> w=PlungeCount-12 ->
  if PlCnt<12, C=0
        btfss STATUS,C     ; if not, get out of ad ops
        goto  ad_done              ;
        movlw 0x20         ; if PWMVal < 0x20, don't sub 10
        subwf PWMVal,w     ; w=PWMVal-20.  if PWMVal<20, C=0, and get out.
        btfss STATUS,C     ;
        goto  ad_done              ;
        movlw 0x9          ;
        subwf PWMVal,f     ; PWMVal = PWMVal - 9
        goto  ad_done
  boostpwm:
        clrf  PlungeCount  ;
        incf  IncCount,1   ;
        movlw 0xa          ;
        subwf IncCount,0   ; check if DecCount is >10
        btfss STATUS,C     ; if not, get out of ad ops
        goto  ad_done              ;
        clrf  IncCount     ; if it is >10, clear DecCount
        movlw 0xff         ;
        subwf PWMVal,0     ;
        btfsc STATUS,Z     ;
        goto  ad_done              ;
        incf  PWMVal,1     ; decrement PWMVal, put back in PWMVal
        incf  LeapCount,f  ; increment PlungeCount
        movlw 0xc          ;
        subwf LeapCount,w  ; check if LeapCount>12 -> w=LeapCount-12 -> if
  LeapCount<12, C=0
        btfss STATUS,C     ; if not, get out of ad ops
        goto  ad_done              ;
```

```
        movlw 0xd0          ; if PWMVal > 0xd0, don't sub 10
        subwf PWMVal,w      ; w=PWMVal-d0.  if PWMVal>d0, C=1, and get out.
        btfsc STATUS,C      ;
        goto  ad_done               ;
        movlw 0x9           ;
        addwf PWMVal,f      ; PWMVal = PWMVal - 9

ad_done:
        comf  PWMVal,0      ; complement PWMVal and store result in w reg
        movwf PWMRem                ;


; if LITEON is high or if AWAKE is low or if BLANK is high,
;  must increment CountLo and CountHi

        btfsc State,LITEON          ; if LITEON is high, jump to countup
        goto  countup               ;
        btfss State,AWAKE ; if AWAKE is low, jump to countup
        goto  countup               ;
        btfsc State,BLANK ; if BLANK is high, jump to countup
        goto  countup               ;
        goto  nocount               ; if neither condition is met, go to nocount

countup
        movlw 0xff          ;
        subwf CountLo,0     ; W=CountLo-255.  Z=1 if CountLo=255
        btfss STATUS,Z      ; if Z is clear, skip incrementing CountHi
        goto  lo_only               ;
        incf  CountHi,1     ;
lo_only
        incf  CountLo,1     ;
        incf  BlankCnt,1    ;

nocount

; if CHECKING is high, increment CheckCnt
        btfss State,CHECKING        ;
        goto  checklo               ;
        incf  CheckCnt,1    ;
; check if CHECKING period (one second) is over - if so, clear CHECKING,
;  and if PWMVal is higher (meaning it got darker in the second since a pulse
was sent),
;  then send another pulse
        movlw 0x64          ; 100 in hex
        subwf CheckCnt,0    ; w = CheckCnt-100.  C=0 if CheckCnt<100
        btfss STATUS,C      ; if C=1, go to check PWMVal against StoredPWM
        goto  checklo               ;

        btfss State,LITEON          ; if LITEON is high, do the "want light on"
version
        goto  liteonlo      ;

; if LITEON is high, do the following
        movf  PWMVal,0      ; move PWMVal into w
        subwf StoredPWM,0 ; w = StoredPWM-PWMVal.  C=0 if StoredPWM<PWMVal
        btfss STATUS,C      ; if C=1 (got lighter), don't send pulse again
```

```
        bsf    PORTC,PULSEOUT     ;
        goto   clrcheck     ;

  ; if LITEON is low, do the following .
  liteonlo:
        movf   StoredPWM,0 ; move StoredPWM into w
        subwf  PWMVal,0    ; w = PWMVal-StoredPWM.  C=0 if PWMVal<StoredPWM
        btfss  STATUS,C    ; if C=1 (got darker), don't send pulse again
        bsf    PORTC,PULSEOUT     ;

  clrcheck:
        bcf    State,CHECKING     ;
  ;     bcf    PORTC,TESTP6       ;
        clrf   CheckCnt    ;
  checklo:

  ; check if awake - if not, check if it's time.  if not, go to top.

        btfsc State,AWAKE ; if AWAKE is high, go to other stuff.  if not, check
  counter
        goto   already_awake      ;

  ; check test pin - if low, go to set_awake

  ; old line below, started in 5 minutes
  ;     movlw 0x76         ; corresponds to 118d, timeout of 5 minutes
        movlw 0x23         ; corresponds to 35d, timeout of 90 seconds
        btfss PORTB,2              ;
        goto   set_awake    ;
        subwf  CountHi,0    ;
        btfss  STATUS,Z     ; if CountHi != 35, go to main_loop again
        goto   main_loop    ;

  ;  if time to go awake, set AWAKE, clear timers, and go to top
  set_awake
        bsf    State,AWAKE ; set AWAKE bit
        bsf    PORTB,TESTP13      ; set external test pin 13
        clrf   CountLo             ;
        clrf   CountHi             ;
        goto   main_loop    ;

  already_awake     nop

  ; check PULSEOUT - if set, increment PWCount
        btfss  PORTC,PULSEOUT     ; if PULSEOUT is not set, go to next section
        goto   exit_pulse         ;

        incf   PWCount,1    ;

  ; check if timeout has been reached, if so then clear it
        movlw  0x14         ; move 20d into W
        subwf  PWCount,0    ; if same, Z==1
        btfss  STATUS,Z     ;
        goto   exit_pulse   ;
        bcf    PORTC,PULSEOUT     ;
        clrf   PWCount             ;
```

```
exit_pulse  nop          ;

; check if BLANK is high - if so, ignore PIR and check if it's time to drop
;  BLANK

      btfsc State,BLANK ;
      goto  check_blank ;

; check PIR inputs - if active, clear out CountLo and CountHi

      btfss PORTC,PIRL  ;
      goto  presence
      btfsc PORTC,PIRH
      goto  presence
      goto  quiet

presence
      incf  PresCnt,1   ;

; if PresCnt > 2, perform ops - otherwise, go back to main loop
      movlw 2
      subwf PresCnt,0   ; W = PresCnt-10.  C=0 if PresCnt < 10
      btfss STATUS,C    ; check C, if set then skip goto, otherwise loopback
      goto    main_loop ;

      clrf  CountHi
      clrf  PresCnt

; check if LITEON - if not, check Dark - if dark, make pulse
      btfsc State,LITEON      ; if LITEON is set, jump to top
      goto  main_loop

; check if PWMVal>128 - if so, set LITEON and call pulse program
      movlw 0x80        ;
      subwf PWMVal,0    ; W = PWMVal-0x80.  C=0 if PWMVal<0x80
      btfss STATUS,C    ; if C is set, generate pulse - otherwise, back to top
      goto  main_loop   ;

      call  query_lite  ; test function for the moment @@@

; if RS232OK is low, set LITEON, set pulseout, save PWMVal in StoredPWM, and set
CHECKING
      btfsc State,RS232OK     ;
      goto  rs_set            ;
      bsf   State,LITEON      ;
      bsf   PORTC,PULSEOUT    ;
      movf  PWMVal,0    ;
      movwf StoredPWM   ;
      bsf   State,CHECKING    ;
      goto  main_loop   ;

; if RS232OK is high, check if WORKLIGHT is low, if so then set pulseout and
LITEON.  then go back to top
rs_set:
      clrf  BlankCnt    ;
      bsf   State,BLANK ; setting BLANK here stops oscillations
```

```
        btfsc State,WORKLIGHT    ;
        goto  main_loop    ;
        bsf   State,LITEON       ;
        bsf   PORTC,PULSEOUT     ;
        goto  main_loop    ;

check_blank

        movlw 0xff         ; if BLANK has been high for 2.5 seconds,
        subwf BlankCnt,0   ;  shut it off
        btfss STATUS,Z     ;
        goto  quiet        ;
        bcf   State,BLANK  ;
;       bcf   PORTC,TEST2  ; clear external test pin 5

quiet:
        clrf  PresCnt              ;
; if PIR inputs are inactive, check if CountHi==118.  if so, clear out LITEON
and pulse
        btfss State,LITEON         ;
        goto  main_loop    ;
        movlw 0x8d         ; 141d, equal to 6 minutes
        btfss PORTB,2              ; if test pin is low, load up 2 as test for
CountHi
        movlw 0x2          ;
        subwf CountHi,0    ; if CountHi==141, then Z=1
        btfss STATUS,Z     ;
        goto  main_loop    ;
        clrf  CountHi              ;
        clrf  CountLo              ;
        bcf   State,LITEON         ;
        bsf   State,BLANK  ;
        clrf  BlankCnt     ;
        call  query_lite   ;

; if RS232OK is low, set pulseout, set StoredPWM equal to PWMVal, set CHECKING,
go to top

        btfsc State,RS232OK        ;
        goto  rs_clr               ;
        bsf   PORTC,PULSEOUT       ;
        movf  PWMVal,0     ;
        movwf StoredPWM    ;
        bsf   State,CHECKING       ;
        goto  main_loop    ;

; if RS232OK is high, check if WORKLIGHT is high, if so then set pulseout.  then
go back to top
rs_clr:
        btfsc State,WORKLIGHT    ;
        bsf   PORTC,PULSEOUT       ;
        goto  main_loop    ;

query_lite:

        bcf   State,WORKLIGHT    ;
```

```
; look for key reading pulse, stay until seen
waittillo:
      btfsc PORTB,1              ; read pin 12, RB1
      goto  waittillo    ;
waittilhi:
      btfss PORTB,1              ; read pin 12, RB1
      goto  waittilhi    ;

; reset timer, timer bits are 32us/bit
      clrf  TMR0         ; clear out TMR0 to start timer again

; wait 500 us
wait500:
      movlw 0x10         ;
      subwf TMR0,0               ;  check if TMR0 = 16 (time = 512us)
      btfss STATUS,Z     ;
      goto  wait500              ;  if not yet, check again
      clrf  TMR0         ;  clear timer
; send 0x3a
;  turn pin 6     on for 1666 us,
;           off for 833us,
;           on for 833us,
;           off for 2499us,
;           on for 1666us.
      bsf   PORTC,RS232DRV     ;
wait1:
      movlw 0x33         ;
      subwf TMR0,0               ;  check if TMR0 = 51 (time = 1664us)
      btfss STATUS,Z     ;
      goto  wait1        ;  if not yet, check again
      clrf  TMR0         ;
      bcf   PORTC,RS232DRV     ;
wait2:
      movlw 0x19         ;
      subwf TMR0,0               ;  check if TMR0 = 26 (time = 832us)
      btfss STATUS,Z     ;
      goto  wait2        ;  if not yet, check again
      clrf  TMR0         ;
      bsf   PORTC,RS232DRV     ;
wait3:
      movlw 0x19         ;
      subwf TMR0,0               ;  check if TMR0 = 26 (time = 832us)
      btfss STATUS,Z     ;
      goto  wait3        ;  if not yet, check again
      clrf  TMR0         ;
      bcf   PORTC,RS232DRV     ;
wait4:
      movlw 0x4d         ;
      subwf TMR0,0               ;  check if TMR0 = 78 (time = 2496us)
      btfss STATUS,Z     ;
      goto  wait4        ;  if not yet, check again
      clrf  TMR0         ;
      bsf   PORTC,RS232DRV     ;
wait5:
      movlw 0x4d         ;
      subwf TMR0,0               ;  check if TMR0 = 78 (time = 2496us)
      btfss STATUS,Z     ;
```

```
        goto  wait5         ;  if not yet, check again
        clrf  TMR0          ;
        bcf   PORTC,RS232DRV    ;

; wait for 100us for pin 12 to rise before checking
wait6:
        movlw 0x3           ;
        subwf TMR0,0                ;  check if TMR0 = 3 (time = 96us)
        btfss STATUS,Z      ;
        goto  wait6         ;  if not yet, check again
        clrf  TMR0          ;

; wait for pin 12 to drop low - if it drops for more than 500us, then
;  RS-232 is active.  If it stays low for less than 500us, RS-232 is
;  inactive.
checkp12:
      .btfsc PORTB,1                ; read pin 12, RB1
        goto  checkp12      ;
        clrf  TMR0          ; reset counter to start measuring low pw

        bsf   State,RS232OK     ;
; first time high is seen, check if time > 512us, or 16 counts.  If so, set
RS232OK.
; also check if time = 2912us, or 91 counts.  If it is, sample pin12 and pass to
WORKLIGHT.  Exit.

reading:
        btfss PORTB,1           ;
        goto  p12lo         ;
        movlw 0x10          ;
        subwf TMR0,0                ; W=TMR0-0x10.  If TMR0<10, then C=0
        btfsc STATUS,C      ;
        goto  p12lo         ;
        bcf   State,RS232OK     ;

p12lo:
        movlw 0x56          ;
        subwf TMR0,0                ;
        btfss STATUS,Z      ;
        goto  reading               ;

; sample pin 12, set WORKLIGHT if HIGH
        btfsc PORTB,1           ;
        bsf   State,WORKLIGHT   ;
        clrf  TMR0          ;

; wait for end of signal from GDO
restofrx:
        movlw 0xc8          ;
        subwf TMR0,0                ;
        btfss STATUS,Z      ;
        goto  restofrx      ;
        clrf  TMR0          ;

; send break
       bsf   PORTC,RS232DRV     ;
```

```
break1:
      movlw 0xff          ;
      subwf TMR0,0              ;
      btfss STATUS,Z      ;
      goto  break1        ;
      clrf  TMR0          ;
break2:
      movlw 0x20          ;
      subwf TMR0,0              ;
      btfss STATUS,Z      ;
      goto  break2        ;
      bcf   PORTC,RS232DRV      ;

      retlw 0             ;

            END                 ; directive 'end of program'
```